NUMERIC FUNCTION GENERATORS

J. T. Butler *T. Sasao* *S. Nagayama*

Department of ECE
Naval Postgraduate School
Monterey, CA U.S.A. 93943-5121
Phone:+1-831-656-3299,
Email:Jon_Butler@msn.com

Department of CSE
Kyushu Institute of Technology
Iizuka, 820-8205 JAPAN
Phone:+81-948-24-7275,
Email:sasao@cse.kyutech.ac.jp

Department of CE
Hiroshima City University
Hiroshima, 731-3194 JAPAN
Phone:+81-82-830-1599
Email:s_naga@hiroshima-cu.ac.jp

ABSTRACT

We show the architecture and design of a numeric function generator that realizes, at high speed, arithmetic functions, like $\log x$, $\sin x$, $\frac{1}{x}$, etc.. This approach is general; different circuits are not needed for different functions. Further, composite functions, like $\log(\sin(\frac{1}{x}))$ can be realized as easily as individual functions. A tutorial description of the method is presented, followed by descriptions of the design considerations that must be made. For example, we discuss how circuit complexity increases as the desired approximation error decreases. Also, we discuss enhancements of the basic numeric function generator approach, including higher order polynomial approximations, floating point, and multi-variable implementations.

1. INTRODUCTION

The realization of arithmetic functions like $\sin x$, $\log x$, and $\frac{1}{x}$ with high-speed and accuracy has been an important problem since the beginning of computers. More than 150 years ago, Babbage devised a mechanical computer for computing tables of logarithms and triginometric functions, in his *difference machine*. Although he never completed his machine, one was completed at the The Science Museum in London, U.K. in 1991 using his plans. A second machine was completed and was on display at the Computer History Museum in Mountain View, CA [3]. In the time of Babbage, the critical application was navigation. It has been suggested that sailors lost their lives due to errors in tables used for navigation [7], which, at that time, depended on human calculation.

Fifty years ago, Volder [16] introduced the CORDIC algorithm for computing logarithmic and trigonometric functions. In this iterative algorithm, successively more accurate bits are computed until the desired accuracy is achieved [1]. The advantage of CORDIC is the relatively modest amount of hardware needed [1]. Indeed, it has been used in hand calculators, beginning in 1972 with Hewlett-Packard's HP-35 [2]. The CORDIC algorithm was also used in Intel's 8087 numeric co-processor [13].

By some measures, the CORDIC algorithm is still fast. It may be implemented in a pipeline, where each stage quickly computes one bit of the result. Typically, the latency or number of clocks needed to compute the entire result is large because of the need to compute successively more accurate bits. If the system in which a CORDIC algorithm computation is embedded is itself a pipeline, this may be acceptable. In a hand calculator, computation speed need not be high because of much slower speed by which a human can input digits.

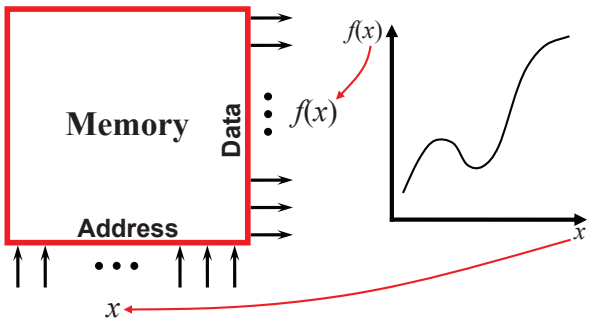

Fig. 1. Single Memory Implementation of $f(x)$.

Thus, CORDIC achieves high-throughput, but has high latency. In order to achieve low-latency and high-throughput, one can use a simple memory, as shown in Fig. 1. In this realization, a binary encoding of x is applied to the address inputs of the memory. The output is the value stored at this address; it is an encoding of the value of the realized function $f(x)$. Table I shows the required memory as a function of the number of bits n used to realize x and $f(x)$. For $n = 8$ and 16 bits, memory size is modest. In this case, the single memory approach is

TABLE I
MEMORY SIZE OF THE LOOKUP TABLE IMPLEMENTATION OF NUMERIC FUNCTION GENERATORS

No. of Bits for x and $f(x)$	No. of Bits in Address	Memory Size in Bytes
8	8	256
16	16	131,072
32	32	1.718×10^{10} (17 Gigabytes)
64	64	1.476×10^{20}
128	128	5.445×10^{39}

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUL 2009		

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Numeric Function Generators	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Electrical and Computer Engineering,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We show the architecture and design of a numeric function generator that realizes, at high speed, arith- metic functions, like log x, sin x, 1 x , etc.. This ap- proach is general; di?erent circuits are not needed for di?erent functions. Further, composite functions like log (sin (1 x)) can be realized as easily as individ- ual functions. A tutorial description of the method is presented, followed by descriptions of the design considerations that must be made. For example, we discuss how circuit complexity increases as the desired approximation error decreases. Also, we discuss en- hancements of the basic numeric function generator approach, including higher order polynomial approxi- mations, ?oating point, and multi-variable implemen- tations.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**4**	

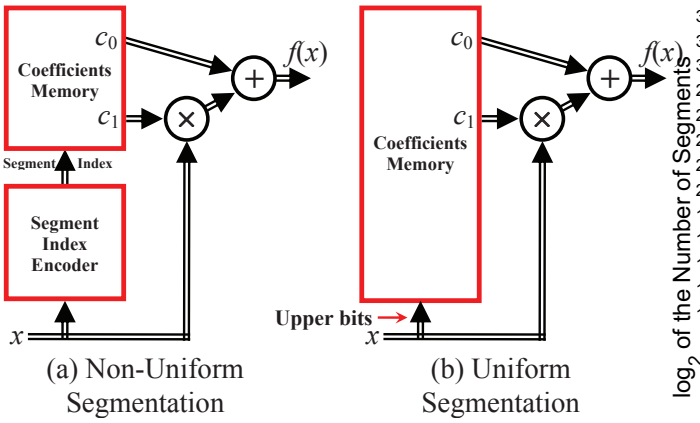

Fig. 2. Architecture of a Numerical Function Generator Using a Piecewise Polynomial Approximation.

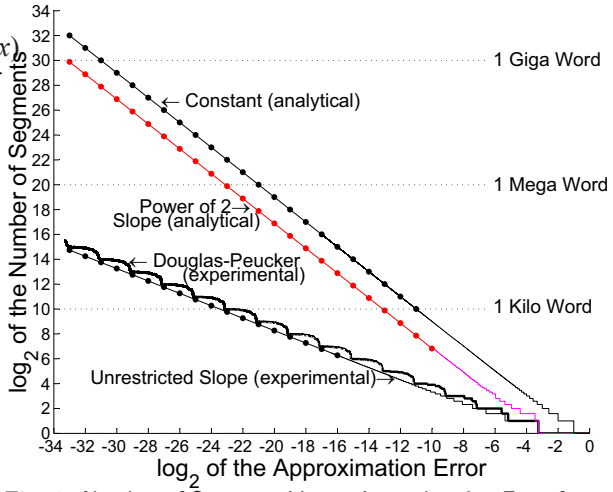

Fig. 3. Number of Segments Versus Approximation Error for $\sin \pi x$, Where $0 \leq x \leq 1/2$.

a reasonable implementation. For $n = 32$ bits, 17 Gigabytes are needed, which is large. For $n = 64$ and 128 bits, the memory size exceeds by a large margin today's technology capabilities.

2. A PIECEWISE LINEAR APPROACH TO REALIZING NUMERIC FUNCTIONS

Fig. 2a) shows the architecture of a numeric function generator that realizes a given numeric function as a piecewise linear approximation. This is is based on a *tabular* approach to realizing numeric functions [6]. The input x drives a *segment index encoder* which produces an index of the segment in which the value of x falls. Within this segment, the function is realized as a line $c_1x + c_0$. The values c_1 and c_0 are outputs of the memory. They drive a circuit that realizes $f(x) = c_1x + c_0$. Sasao, Butler, and Riedel [14] show that the segment index encoder is tractably realized as a look-up table (LUT) cascade.

Fig. 3 shows how the memory size depends with the approximation error for the $sin(\pi x)$ function, where $0 \leq x \leq 1/2$. Plotted vertically is the log_2 of the number of segments versus log_2 of the approximation error. Smaller approximation error values are on the left and larger approximation error values are on the right. The top line, labeled Constant (analytical), corresponds to a constant approximation, in which the approximating line is horizontal. It corresponds to a memory output for c_1 equal to 0. In this case, a multiplier is not needed, which is a source of much delay in the circuit. Note, however, that a large number of segments are needed.

The next line, labeled Power of 2 Slope (analytical), shows the number of segments needed in the case where c_1 is restricted to be a power of 2. In this case, the multiplier is a shift operation. As such, there is some delay, but not as much as with a full multiplier. The number of segments is smaller, but still large.

The third line, labeled Douglas-Peucker (experimental), shows the number of segments associated with the circuit shown in Fig. 2a) when the Douglas-Peucker algorithm [4] is used is determine the segments. This is a heuristic in which segments are determined iteratively. First, one line is used to approximate the whole domain. Then, the point of maximum error is used to partition the domain into two parts, etc.. This process is repeated until the maximum error is not greater than the desired error over the whole domain.

The bottom line, labeled Unrestricted Slope (experimental), shows the number of segments when the segmentation is optimum. This shows that the Douglas-Peucker algorithm is close to optimum, while the constant slope and power of 2 slope are far from optimum.

The circuit in Fig. 2a) is said to realize a *non-uniform segmentation*. Fig. 2b) shows the architecture of a numeric function generator that realizes a given numeric function as a piecewise linear approximation in which all of the segment widths are the same. This architecture realizes *uniform segmentation*. Normally, a segment index encoder would also be used in this circuit. However, we will choose the (uniform) width to be some power of 2. In this case, the segment index encoder can be omitted and the most significant bits of x are applied to the memory address input. Since a linear approximation is still involved, the circuit realizing $c_1x + c_0$ remains.

3. NON-UNIFORM VERSUS UNIFORM SEGMENTATION

It is shown that, for **nonuniform** segmentations

Theorem 1: [5] *Consider a piecewise linear approximation of f on the domain $[a, b]$ that is accurate to within ε, using a piecewise linear segmentation. Let f be three times continuously differentiable on $[a, b]$. Then, $s(\varepsilon)$, the number of segments in an optimum non-uniform segmentation of $[a, b]$, satisfies the fol-*

lowing asymptotic approximation:

$$s(\varepsilon) \sim \frac{c}{\sqrt{\varepsilon}}\ , (\varepsilon \to 0), \tag{1}$$

where

$$c = \frac{1}{4}\int_a^b \sqrt{|f''(x)|}dx. \tag{2}$$

Further, it is shown that, for **uniform** segmentations

Theorem 2: [5] *Consider a piecewise linear approximation of a function $f(x)$ on the domain $[a, b]$ with a specified approximation error ε or less using uniform segmentation. Let the absolute value of the second derivative $|f''(x)|$ of $f(x)$ on the domain $[a, b]$ be finite. Then, the number of segments s is*

$$s \sim \frac{c}{\sqrt{\varepsilon}}, \tag{3}$$

where

$$c = \frac{(b-a)\sqrt{|f''|_{\max}}}{4}, \tag{4}$$

where $|f''|_{\max}$ is the maximum of the absolute value of $f''(x)$ over the domain $[a, b]$.

For non-uniform approximation, the number of segments $s(\varepsilon)$ depends on the integral of the second derivative over the interval of approximation, which is similar to an average. The theorem requires that the function $f(x)$ be three-times differentiable; this implies the second derivative is integrable. For uniform approximation, the number of segments depends on the maximum value of the second derivative. These values can be quite different, depending on the function.

Table II shows the number of segments for 14 numeric functions, as computed from Theorems 1 and 2 and for the two types of segmentation, non-uniform (1) and uniform (3), and for four precisions, 8, 16, 32, and 64 bits. For 64 bit precision, all functions require a very large memory size, while 32 bit precision yields feasible realizations, except for three functions. For example, for $\sqrt{x}$, the number of segments needed in a uniform segmentation is much larger than in a non-uniform segmentation. This is due to a large absolute value for the second derivative near $x = 0$. Indeed, for all four precisions, uniform segmentation requires many more segments than non-uniform segmentation. Similarly, $\sqrt{-\ln(x)}$ and $-(x\log_2 x + (1-x)\log_2(1-x))$ require many more segments using uniform segmentation than for non-uniform segmentation.

In comparing the two types of segmentations, it is necessary to account for the complexity of the segment index encoder. We know of no analytic way to measure its complexity. However, experimental results [15] show that, with uniform segmentation, the $\ln x$, $\sqrt{x}$, and $1/x$ functions cannot be implemented on an Altera Stratix EP1S20F484C5 FPGA, while a non-uniform implementation can.

4. EXTENSIONS OF THE BASIC NFG

Higher Order Approximating Polynomials

A function that is close to linear is efficiently approximated by a linear function, $c_1x + c_0$. From Table II, $\frac{1}{1+e^{-x}}$ can be seen to be linear because of the relatively few number of segments needed for both non-uniform and uniform approximations. However, other functions are highly non-linear. This suggests that there is an advantage to using quadratic, cubic, and higher order polynomials. It is known that quadratic polynomial approximations can drastically reduce the number of segments to as little as 4% of the segments needed in a linear approximation [8].

A disadvantage of higher order polynomials is the need for additional multipliers to realize the higher powers of x. This uses significant FPGA resources and has larger delay. Indeed, it is known [10] that linear and quadratic polynomials yield the highest efficiency designs.

Floating Point

We have discussed so far only fixed point representations. This restricts the domain, as well as the application. Nagayama, Sasao, and Butler [12] have shown the use of edge-valued decision diagrams in the design of floating point numeric function generators for monotone elementary functions.

Multi-Variable Functions

A multi-variable function depends of two or more variables. For example, the multi-variable function $f(x, y) = \sqrt{x^2 + y^2}$ is used in converting from cartesian to polar coordinates. Such a function can be realized by combining three single-variable functions, two realizing α^2 and one realizing $\sqrt{\beta}$. A more efficient approach is to realize it directly using rectangles to approximate a surface [9], which is analogous to the approach described above for single-variable functions. This approach yields a 58% memory size reduction and a 39% delay time reduction over the approach in which a number of single-variable functions are used [9]. A further simplification can be achieved by observing that this function is symmetric, i.e. $f(x, y) = f(y, x)$ and that, effectively only one-half of the surface need be realized [11].

5. CONCLUDING REMARKS

There is a long history of realizing numeric functions, like $\sin x$ by computer. Today's FPGAs provide large amounts of flexible logic at reasonable cost. We propose the use of linear and higher-order

TABLE II
NUMBER OF SEGMENTS FOR NON-UNIFORM AND UNIFORM SEGMENTATION FOR 8, 16, 24, AND 32 BIT PRECISION [5].

Function $f(x)$	Interval x	Non-Uniform 8	16	32	64	Uniform 8	16	32	64
2^x	$[0,1)$	4	75	19,195	1.26×10^9	6	89	22,717	1.49×10^9
$1/x$	$[1,2)$	4	75	19,195	1.26×10^9	8	128	32,773	2.15×10^9
$\sqrt{x}$	$[0,2)$	10	216	55,109	3.61×10^9	8,206	5.38×10^8	2.31×10^{18}	4.26×10^{37}
$1/\sqrt{x}$	$[1,2)$	3	50	12,772	8.37×10^8	5	79	20,066	1.32×10^9
$\log_2(x)$	$[1,2)$	4	75	19,228	1.26×10^9	7	109	27,833	1.82×10^9
$\ln x$	$[1,2)$	3	63	16,062	1.05×10^9	6	91	23,171	1.52×10^9
$sin(\pi x)$	$[0,\frac{1}{2})$	5	109	27,759	1.82×10^9	9	143	36,397	2.39×10^9
$cos(\pi x)$	$(0,\frac{1}{2})$	5	109	27,759	1.82×10^9	9	143	36,397	2.39×10^9
$tan(\pi x)$	$[0,\frac{1}{4})$	4	73	18,583	1.22×10^9	9	143	36,397	2.39×10^9
$\sqrt{-ln(x)}$	$[\frac{1}{256},\frac{1}{4})$	10	216	55,248	3.62×10^9	157	2,507	641,600	4.20×10^{10}
$tan^2(\pi x)+1$	$[0,\frac{1}{4})$	7	153	38,927	2.55×10^9	18	285	72,793	4.77×10^9
$-(x\log_2 x+(1-x)\log_2(1-x))$	$(0,1)$	16	342	87,437	5.73×10^9	136	34,787	2.28×10^9	9.79×10^{18}
$\frac{1}{1+e^{-x}}$	$[0,1)$	1	20	5,096	3.34×10^8	2	28	6,989	4.58×10^8
$\frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}}$	$[0,\sqrt{2}]$	3	53	13,458	8.82×10^8	6	81	20,696	1.36×10^9

approximations to realize general numeric functions. One advantage of this is that a wide range of functions can be synthesized in an architecture that is similar for each function. We have discussed the extension of this to floating point and multi-variable functions. We believe that, as technology improves, there will be further opportunities to research this interesting topic. We conclude with the following:

Open Question: Does there exist an analytical quantification of the memory size needed for the segment index encoder that depends on the approximation error and function properties?

REFERENCES

[1] R. Andrata, "A survey of CORDIC algorithms for FPGA based computers," *Proc. of the 1998 ACM/SIGDA Sixth Inter. Symp. on Field Programmable Gate Arrays (FPGA '98)*, pp. 191-200, Monterey, CA, Feb. 1998.

[2] D. Cochran, "Algorithms and accuracy in the HP-35," *Hewlett-Packard Journal*, Vol. 23, pp. 10-11, June 1972.

[3] Computer History Museum website - http://www.computerhistory.org/babbage/

[4] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a line or its caricature," *The Canadian Cartographer*, Vol. 10, No. 2, pp. 112-122, 1973.

[5] C. L. Frenzen, T. Sasao, and J. T. Butler, "The tradeoff between memory size and approximation error in numeric function generators based on lookup tables," preprint.

[6] H. Hassler and N. Takagi, "Function evaluation by table look-up and addition," *Proc. of the 12th IEEE Symp. Comp. Arith. (ARITH'95)*, Bath, pp. 10-16, July 1995.

[7] J. L. Latham, *Carry On, Mr. Bowditch*, Houghton Mifflin Company, New York, NY, 1955 and 1983, http://www.commonsensepress.com/sample_book_studies_pdfs/tan_tbook_study_sample.pdf.

[8] S. Nagayama, T. Sasao, and J. T. Butler, "Compact numerical function generators based on quadratic approximations: Architecture and synthesis method,", *IEICE Trans. Fund.*, Vol. E89-A, No. 12, pp. 3510-3518, Dec. 2006.

[9] S. Nagayama, J. T. Butler, and T. Sasao "Programmable numerical function generators for two-variable functions," *10th EUROMICRO Conf. on Dig. Sys. Des. Arch., Methods and Tools, (DSD 2008)*, pp. 891-898, 1-5 Sept. 2008.

[10] S. Nagayama, J. T. Butler, and T. Sasao "Design method of numerical function generators based on polynomial approximation for FPGA implementation," *10th EUROMICRO Conf. on Dig. Sys. Des. Arch., Methods and Tools, (DSD 2007)*, pp. 280-287, 27 - 31 Aug. 2007.

[11] S. Nagayama, T. Sasao, and J. T. Butler, "Programmable architectures and design methods for two-variable numeric function generators," *preprint*, 2009.

[12] S. Nagayama, T. Sasao, and J. T. Butler, "Floating-point numerical function generators using EVMDDs for monotone elementary functions," *Proc. of the 39th Inter. Symp. on Multiple-Valued Logic*, pp. 349-355, May 21-23, 2009.

[13] R. Nave, "Implementation of transcendental function on a numerics processor," *Microprocessing and Microprogramming*, Vol. 11, pp. 221-225, 1983.

[14] T. Sasao, J. T. Butler, and M. D. Riedel, "Applications of LUT cascades to numerical function generators," *Proc. of the 12th Workshop Synthesis and System Integration of Mixed Information Technologies (SASIMI '04)*, pp. 422-429, Oct. 2004.

[15] T. Sasao, S. Nagayama, and J. T. Butler, "Numerical function generators using LUT cascades," *IEEE Trans. on Comp.*, Vol. 56, No. 6, pp. 826-838, June 2007.

[16] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Comput.,* Vol. EC-8, No. 3, pp. 330-334, Sept. 1959.